Principis de modelització en un desenvolupament de software

(La difícil tasca d'identificar. Número 1)

Josep M. Merenciano

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

meren@lsi.upc.edu

Març de 2012

Resum

Tot desenvolupament de software es basa en la construcció i transformació de models. Un model és una construcció arbitrària que és útil de cara a uns propòsits determinats. L'arbitrarietat dels models, però, té uns límits que no es poden traspassar. Aquí analitzem quins són aquests límits en termes de l'estructura del model i de la seva relació amb la realitat modelitzada. En el cas dels models emprats en el desenvolupamnt de software cal afegir unes restriccions addicionals: quines unitats de software admetem, com es comuniquen entre elles, i quines propietats fonamentals han de mantenir.

Abstract

All software development is based on the construction and transformation of models. A model is an arbitrary artifact that is useful for a specific purpose. The arbitrariness of the models, however, has limits that can not be crossed. Here we analyze what these limits are, in terms of model structure and its relation to the modeled reality. In the case of the models used in software development we must add some additional restrictions: which kind of software units are supported, how they communicate, and what fundamental properties should be maintained.

1 Gènesi

Ús de coneixement implícit. En la meva experiènca en l'ensenyament (des del 2005) de les tècniques i mètodes de l'enginyeria del Software aviat em vaig adonar de com els professionals prenem algunes decisions tant insconscientment que ens sembla inconcebible que algú altre (els estudiants) pugui pendre decisions diferents. En l'anàlisi del motiu d'aquesta discrepància vaig dscobrir que molts cops darrera hi havia un coneixement per part del docent o professional que era desconegut per l'estudiant. El sorprenent del cas era que aquest coneixement, fruit d'anys d'experiència i de múliples lectures, era encapsulable en alguns principis simples, tant d'enunciació com d'aplicació.

L'Enginyeria del Software encapsula el coneixement. Però aquesta és justament l'essència de l'Enginyeria del Software. La disciplina, l'art, l'habilitat i la professió d'adquirir i aplicar coneixements científics, matemàtics, econòmics, socials i pràctics, en el desenvolupament de software[1] és tant la capacitat de reproduir desenvolupaments com la capacitat d'encapsular l'experiència prèvia, pròpia o d'altri, de transmetre-la i d'usar el coneixement rebut a través de l'experiència d'altres.

Transmissió errònia de coneixement. Des d'aquest punt de vista, la discrepància entre el docent i el discent no és deguda a una mancança del discent, ans a una mancaça del docent. L'origen de la discrepància rau en la incapacitat del docent d'haver transmès al discent la seva experiència; el docent de l'Enginyeria del Software falla en no usar ell mateix les tècniques i mètodes que pretén ensenyar.

Propòsit d'esmena. Arribats a aquesta conclusió només hi havia un camí a seguir. Calia detectar les discrepàncies i per cadascuna analitzar quin era el coneixement emprat implícitament pel professional, explicitar-lo, i encapsular-lo en uns pocs principis simples. I això fer-ho extensiu en tot el contingut de l'assignatura Enginyeria del Software:Disseny[2] de la que l'autor n'era responsable. En alguns casos l'explicitació del coneixement ha estat simple, d'altres força més complex. Un cop explicitat el coneixement sovint els principis apareixien per si sols, bé per tractar-se de principis fortament coneguts, bé per tractar-se de principis extensament usats tot i que potser no explicitats o sense un nom d'ús universal. De tot plegat en sorgí un llibre (inèdit, però accessible pels estudiants de l'assignatura) on a mesura que es va avançant en un desenvolupament es van analitzant les diferents decisions possibles, els seus avantatges i inconvenients, com les anàlisis es poden encapsular en principis, i com l'ús d'aquests principis simplifica el desenvolupament.

La problemàtica dels identificadors. El què, el perquè, el com i el quan dels identificadors ha esdevingut un obstacle feixuc. Poca cosa hi ha en la biblio-

[1] Viquipèdia. Entrada "'Enginyeria".

[2] Enginyeria Tècnica en Informàtica de Gestió, Escola Politècnica Superior d'Enginyeria de Vilanova i la Geltrú. El darrer cop que es donà aquesta assignatura, per extinció del pla, fou la primavera de 2012.

grafia aplicable al context que m'interessava (visió des de l'especificació i des del disseny, fugint d'implementacions concretes com les bases de dades o l'orientació a objectes), i en canvi són molts els dubtes que els apareixen als estudiants. Per resoldre la problemàtica vaig introduir un capítol en el llibre usat com a material docent que desenvolupés els principis pertinents. Però les premisses emprades en aquests principis estaven farcides de coneixement implícit, la qual cosa les feia incomprensibles als estudiants. Calia explicitar aquest coneixement i presentar nous principis. La bola de neu s'anà fent gran i cada cop més tècnica. El resultat queda lluny del nivell dels estudiants als qui originalment anava destinat.

Informes de recerca amb origen en material docent. El material que tot i ser originalment docent és més de recerca que de transmissió docent l'he refet en termes d'informes de recerca, tot i mantenint l'estil de la redacció. El resultat és un híbrid: el contingut és més de recerca que docent, però l'estil i l'estructura el permet emprar com a material de consulta en cursos superiors.

Un apunt sobre l'estil. Com a material docent un text ha de permetre múltiples lectures: una ràpida que ens permeti situar dins la problemàtica, una pausada d'aprenentatge del contingut, lectures ràpides d'estudi i repàs, ús com a material de referència, accès ràpid a contingut antic relacionat amb el contingut nou, etc. Per aquest motiu aquest text conté diferents índexs, múltiples referències creuades, repeticions volgudes de contingut (amb el mateix text o amb text alternatiu, amb ús de diferents registres o nivells de formalisme,...), i els paràgrafs es presenten com un unitat semàntica. Aquesta darrera afirmació significa que cada paràgraf introdueix una sola idea o conseqüència; i aquesta idea es pot plasmar o resumir en el títol que encapçala el propi paràgraf. El resultat és un text més llarg del necessari per a la simple exposició, potser més feixuc a voltes, però que facilita gran part de les múltiples lectures exigides a un material docent.

2 Contingut

El contingut d'aquest informe. En aquest informe presentem el marc teòric bàsic al voltant del qual volten els altres informes de recerca que constitueixen la sèrie *La difícl tasca d'identificar*. En concret, els principis que defineixen quins són els models admissibles, així com els noms usats per a referir-nos a elements d'un model; però també les restriccions que imposem a les unitats software.

Comunicació entre models. En els propers informes analitzem en primer lloc la necessitat de comunicació entre dos models (o entre un model i una realitat). L'abstracció apareix com un element fonamental per tal que la comunicació sigui factible. L'aparició de les abstraccions i les seves realitzacions introdueix la necessitat de fer una comunicació estratificada: cal comunicar tant l'abstracció com la realització dins d'aquesta abstarcció.

Models infinits. La comunicació de models amb infinits elements requereix una anàlisi a part.

Noms dels elements. Un cop comprès el mecanisme de comunicació entre models comencem a analitzar els noms dels elements d'aquests. Analitzem qüestions com: Tota abstracció i tota realització necesita un nom? Quins són els noms de les associacions? Quins són els noms de les especialitzacions? Quins són els noms dels conceptes associatius? Però per tal de respondre aquestes qüestions cal endinsar-nos en l'essència dels diferents elements que tenim en un model conceptual: concepte, especialització, associació i concepte associatiu.

Bateig. Després d'haver analitzat quins són els elements que necessiten nom, i quins d'aquests l'obtenen mitjançant principis d'inferècnia, queda batejar els elements que són la base d'aquestes inferències. Els noms que introduïm són elements del model i per tant s'han de regir pels principis de modelització.

Els identificadors. Finalment presentem els identificadors i quin és el seu paper en un model.

3 Principis de modelització

3.1 Model i 'realitat'

Definció. Model. *Representació de les propietats rellevants d'allò que volem estudiar.*

Focalització, abstracció i simplicitat. Tot model es basa en els principis d'abstracció, simplicitat i focalització. El model només intenta expressar allò que ens interessa pel nostre estudi (*focalització*); ho fa en els termes més genèrics possibles, oblidant-se de les circumstàncies o contextos particulars (*abstracció*); i cercant sempre la representació més simple (*simplicitat*).

Multiplicitat de models. Una mateixa "realitat" tindrà tants models com objectius tinguem sobre aquesta realitat: cada model focalitza sobre objectius diferents.[3]

Espill. En la construcció del model ens emmirallem en la "realitat" modelitzada. és a dir, *intentem* que cadascun dels conceptes, idees, objectes, etc. es corresponguin *un a un* amb conceptes, idees, etc, de la "realitat" modelitzada.

Model especular. Sovint parlem de *model especular* per referir-nos a un model que segueix el principi de l'*Espill*.

> **Exemple 1 (Quadre a l'oli)** *Un quadre a l'oli és una representació d'allò que estem pintant. Si pintem una casa amb el seu jardí, la "realitat"* casa *la modelitzarem amb un element del quadre diferent a l'element que usem per modelitzar la "realitat"* arbre*. En el quadre tenim diferents formes, de diferents colors, i cadascuna d'elles és la representació d'algun dels objectes que estem pintant. El quadre és una visió especular d'allò que estem pintant.*

> **Exemple 2 (Pintura puntillista)** *Aquest és un exemple de model no especular (no seguim el principi de l'*Espill*). La "realitat" que volem modelitzar només la podem copsar si considerem el quadre en la seva totalitat. No hi ha cap forma, ni cap color que representi la casa; no hi ha cap forma ni cap color que representi els arbres. Només quan contemplem el quadre a certa distància, l'amalgama de punts de color, d'aparença totalment caòtica, s'ordenen i presenten una visió on apareix la casa i els arbres.*

[3] Posem "realitat" entre cometes perquè res impedeix fer models de creacions abstractes, com per exemple un altre model. Així distingim entre la "realitat" modelitzada i el model que l'expressa.

3.2 El nom i la cosa

Franquícia. En el moment de construir un model *podem* usar per anomenar els elements del model noms que es corresponguin amb elements de la "realitat" modelitzada, sempre i quan hi hagi un cert correlat semàntic entre els elements que comparteixen nom.

Denominació consistent amb la semàntica de la franquícia. El préstec o lloguer d'un nom significa que aquest nom passa a tenir més d'un significat: anomena un element del model, però també anomena un element de la "realitat" modelitzada. Per evitar mals entesos el préstec només es pot fer sota el *contracte de la franquícia*: si usem un nom de la "realitat" en un element del model és perquè aquest element d'alguna manera modelitza alguns dels aspectes d'allò que en la "realitat" modelitzada es diu de la mateixa manera. En d'altres paraules: la denominació ha de ser semànticament consistent.[4]

Franquícia obligada. Si en el model podem usar un nom de la "realitat" modelitzada (principi de la *Franquícia*), l'hem d'usar.

Continuïtat de denominació. La franquícia obligada té una conseqüència important: existeix *continuïtat* en els noms dels diferents nivells de modelització o representació, és a dir, tenim un mateix nom per tots els elements que en darrera instància modelitzen la mateixa "realitat"

Referent únic. L'homonímia és sinonímia. És a dir, tots aquells noms idèntics (homònims) necessàriament s'han de referir a una mateixa "realitat" (són sinònims), independentment d'on s'usin, si en la "realitat" modelitzada, si en el model, si en el model del model, etc.

Concreció, claredat, no ambigüitat. El principi del *Referent únic* recolza en els objectius de concreció, claredat i no ambigüitat. El model és un mitjà per expressar i comunicar un coneixement. Si admetem que un mateix nom pot significar "realitats" diferents (subversió del principi del *Referent únic*) llavors ens cal afegir el context desambiguador necessari per fer la comunicació efectiva (amb la conseqüent pèrdua de claredat o concreció).

[4] Una franquícia de sabates només pot vendre sabates, i no pot pas vendre plàtans. En usar el nom de la franquícia ens comprometem a fer servir aquest nom només per allò que el contracte de la franquícia permet.

3.3 Modelització continua

Modelització continua. Els noms i elements usats en el model s'han de correspondre a noms i elements de la "realitat" modelitzada.

Modelització especular amb franquícia obligada. La *franquícia obligada* ens diu que *hem* de reusar noms allà on sigui possible; i la modelització *especular* fa que la reutilització dels noms sigui possible a tot element del model (o quasi). Així, la *Modelització continua* és el resultat de la concurrència dels principis de l'*Espill* i de la *Franquícia obligada*.

Continuïtat entre la "realitat" i el model. En aplicar el principi de la *Modelització continua* obtenim que el model i la "realitat" modelitzada tenen estructures similars (o això hem intentat), i pels elements que es corresponen (un és la representació de l'altre) usem un mateix nom. D'aquesta manera la modelització no "trenca" ni crea una fissura en el nostre coneixement de les coses.

Traçabilitat de la modelització. Aquesta continuïtat entre el model i la "realitat" es veu reforçada amb l'aplicació del *Referent únic*: donat un nivell de modelització qualsevol, podem arribar a la "realitat" modelitzada per un nom del model seguint un *únic camí* que segueix la traça d'aquest nom a través de tots els nivells de modelització. Aquest camí únic és el que anomenem la *traça de la modelització* d'aquest nom.

> **Exemple 3 (Models en el desenvolupament)** *La classe* `Client` *és la representació a nivell d'implementació del component* `Client`*, que ha aparegut en el disseny com a representació software del concepte* `Client` *que apareix en l'especificació del problema; el qual, al seu torn, és la manera d'expressar la idea de* `Client` *que tenen els usuaris del sistema.*
>
> *La* traça de la modelització *de* `Client` *és la seqüència (única) que passa per una idea, un concepte, un component i una classe d'un llenguatge OOP, tots ells amb el nom* `Client` *(i no hi ha cap nom* `Client` *que no estigui dins d'aquesta traça).*

4 Principis de desenvolupament de software

Es tracta de principis usats en les diferents etapes del desenvolupament d'un sistema software. La diferència amb els principis de modelització és que ara els models usats es corresponen a conceptes del software.

4.1 Encapsulament

Encapsulament. Cada unitat de la solució software té uns límits ben definits: sabem on comença, on acaba, i amb qui interactua.

Exemple 4 (Goto) *En* BASIC *les subrutines només es diferencien perquè en el seu fil d'execució troben un* `return`*, i la seva crida es fa amb un* `gosub` *enlloc d'un* `goto`*. Imaginem un codi on sembla que hi ha una rutina A de la línia 100 a la 200 (tenim un* `return` *a la línia 200); i una altra subrutina B entre les línies 300 i 400. Suposem que a la línia 350 hi ha un* `goto 180` *i que en la línia 10 fem* `gosub 320`*. Llavors:*

- *Hem entrat a B per un punt intermedi, no pas per la seva primera línia*
- *A la línia 350 entrem a A, però no pas com a subrutina*
- *A la línia 200 trobem el* `return` *que en aquest cas ens fa el retorn de la crida de la línia 10*

Llavors, quins són els límits de la subrutina B? I els de A? En BASIC *els límits són inexistents; no hi ha encapsulament.*

4.2 Ocultació

Ocultació. Cada unitat de la solució software oculta aquells detalls de la seva construcció que són susceptibles de canvis.

Dues perspectives: ús i construcció. El principi de la *Ocultáció* introdueix la idea que tota unitat de la solució software es pot veure des de dues perspectives diferents: la perspectiva de l'*ús* i la perspectiva de la *construcció*.

Dos actors: client i servidor. Les unitats de la solució software s'usen executant-les d'alguna manera, per exemple, mitjançant una *crida* realitzada des d'una altra unitat de la solució software. En usar una unitat estem demanant que ens ofereixi un servei; l'usuari és el *client* d'aquest servei; la unitat usada és el *servidor* del servei.

Perspectiva i coneixement. El principi d'*Ocultació* ens diu que el coneixement que de la unitat software en té un client, és diferent de la que en té el servidor (això és, la pròpia unitat software usada). Com a servidors tenim tot el coneixement possible; com a clients hi ha informació que ens queda amagada, inaccessible.

Afectació local dels canvis. El principi d'*Ocultació* demana que allò que s'amagui sigui allò que és susceptible de canvi. D'aquesta manera els canvis produïts en el servidor (és a dir, en la construcció de la unitat software) no afecten el client. Com a conseqüència, els canvis no es propaguen, sinó que només afecten localment.

4.3 Interfície

Definció. Interfície.[5] *Descripció de com usar una unitat de la solució software*

En la interfície s'expliciten totes les maneres possibles d'usar la unitat de software. N'és el manual d'instruccions.

> **Exemple 5** *En el món de la codificació C considerem les unitats software anomenades C-funcions. La interfície d'una C-funció ha d'expressar quin és el nom exacte d'aquesta (que serà el nom usat per fer les crides) i quina informació manipula. En concret caldrà donar la seqüència de tipus dels arguments necessaris, i la indicació de si la informació és entrant o sortint.*
>
> *Cal observar com els noms (ficticis) dels paràmetres no formen part de la interfície (tot i que si que es poden incloure en la capçalera C dins el fitxer .h), ja que no donen cap mena d'informació sobre l'ús de la C-funció.*
>
> *En el cas del C, la indicació de si la informació és entrant o sortint a la C-funció no és explícita. En alguns casos es pot extreure del tipus de l'argument; en d'altres de la presència de la paraula reservada* `const`*.*

Interfície obligada. L'ús de cada unitat de la solució software està perfectament delimitat i explicitat.

El principi de la *Interfície obligada* el que ens diu és que tota unitat de la solució software té una *interfície* definida.

4.4 Contractes

Insuficiència de les interfícies. La preseència d'una *interfície* diu com es pot usar la unitat software, però no diu res dels contextos on es pot usar, ni de quins serveis són els que ofereix. Per a això necessitem els *contractes*.

Definció. Contracte. *Descripció explícita del comportament i de l'ús (com usar i en quins contextos) una unitat de la solució software.*

Manual del von ús. En el contracte s'expliciten les situacions correctes d'ús de la unitat de software, i què és el que hom espera com a resultat de l'ús d'aquesta unitat. El contracte és el manual del bon ús i el manual del quan i el perquè ens cal l'ús.

---

[5] No s'ha de confondre la *interfície* com a descripció explícita de com s'ha d'usar una unitat software amb conceptes homòlegs com per exemple les `interface` de llenguatges tipus Java, o els mecanismes d'interacció gràfica amb l'usuari extern del sistema (Interfície Gràfica d'Usuari, IGU). Potser un dels conceptes que més s'hi assembla del mon de la programació i codificació són les capçaleres del C.

Exemple 6 *La C-funció* `facturació(Empresa)` *es preocupa de fer la facturació del client indicat amb l'argument de tipus* `Empresa`*. La* `facturació()` *només es pot realitzar dins la primera setmana de cada mes; i només per a aquells clients que tinguin albarans pendents de facturar. El resultat de llançar la petició és que al client en qüestió se li ha emès una factura per tots els albarans del mes anterior.*

Hem respost les següents qüestions:

Què fa. *Emetre una factura per tots els albarans del mes anterior del client indicat*

Significat dels arguments. *L'argument rebut ha de ser una* `Empresa`*, i representa el client al qui volem facturar*

Contextos vàlids. *Primera setmana de cada mes. El client sobre el que es vol facturar ha de tenir albarans pendents*

Resultat. *S'ha emès la factura amb els albarans pendents del mes anterior, els quals deixen d'estar pendents*

Contractes PRE/POST. Un tipus de contracte molt usat és el que expressa els contextos en forma de *precondicions (PRE)* i *postcondicions (POST)*.

Definció. Contracte PRE/POST. *Contracte que s'expressa mitjançant els següents elements:*

- *Interfície*
- *Explicació semàntica del comportament i del significat de la informació manipulada per la interfície*
- *Condicions contextuals d'ús correcte (PRE)*
- *Condicions resultants d'un ús correcte (POST)*

Lectura d'un contracte PRE/POST. La interfície diu com usar la unitat de software. La semàntica expressa quina és la funcionalitat d'aquesta unitat, i quin paper hi juga cadascun dels elements d'informació involucrats. Les condicions PRE i POST expressen els contextos d'ús correcte, i les condicions que s'obtenen sota un ús correcte.

Exemple 7 (Contracte PRE/POST)

Interfície. `facturacio(Empresa)`

Descripció. *Genera una factura amb els albarans del mes anterior del client indicat; i marca aquests albarans com a no pendents.*

Arguments. *Client sobre el que volem fer la facturació.*

PRE. *El client existeix dins el sistema i té albarans pendents del mes anterior. Estem dins la primera setmana del mes.*

POST. *S'ha generat la factura pels albarans pendents del mes anteriors, corresponents al client; el client no té albarans pendents del mes anterior.*

4.5 Desenvolupament sota contracte

Desenvolupament sota contracte. El client només s'ha de preocupar d'assegurar les PRE; el servidor només s'ha de preocupar d'assegurar les POST.

Divisió de responsabilitats; confiança mútua. El principi del *Desenvolupament sota contracte* proposa una divisió de responsabilitats fonamental, basada en una *doble confiança*: el servidor ha de confiar en què el client només requereix els seus serveis sota els contextos apropiats; i al seu torn el client ha de confiar que el servidor, sempre i quan els serveis se li demanin en els contextos apropiats, es comportarà oferint el que li exigeix el contacte.

4.6 Caixa blanca i caixa negra

Definció. Caixa blanca. *Unitat software* encapsulada *i amb* interfície *ben definida.*

Origen de la nomenclatura. És una caixa perquè els límits estan ben perfilats (principi d'*Encapsulament*), i els usos estan explicitats en la *interfície* (només hi podem accedir per la tapa o interfície). És blanca[6] perquè podem obrir la tapa i veure el seu interior, o alternativament mirar a través de les seves parets transparents.

Ocultació total. El client només coneix els *usos* de la unitat software.

Coneixement limitat a la interfície. El principi de la *Ocultació total* diu que el client coneix la *interfície* de la unitat, però en desconeix les seves interioritats.

Definció. Caixa negra. *Caixa blanca amb ocultació de* tot *el seu interior (l'únic visible és la* interfície*).*

[6] Seria millor dir que és *transparent.*

Principis que porten a la caixa negra. És a dir, una *caixa negra* és una unitat de software construïda segons els següents principis:

1. *Encapsulament*: La unitat està *encapsulada*
2. *Interfície obligada*: Els seus usos estan ben definits
3. *Ocultació total*: La *interfície* és l'únic que el client coneix de la unitat

> **Definció. Component.** Caixa negra*, amb* contracte *explícit, usada com a unitat d'assignació de responsabilitats en el disseny.*

4.7 Desenvolupament amb caixes negres

4.7.1 Independència entre l'ús i la construcció

Dualitat client/servidor fins a l'extrem. L'ús de les *caixes negres* com a única unitat en el desenvolupament d'una solució software porta a l'extrem la doble perspectiva client/servidor.

Ús amb desconeixement. Els clients només coneixen la interfície de la caixa, i per tant tot desenvolupament en el que s'usi la caixa es pot fer (de fet no hi ha més remei que fer-ho així!!) sense saber com és per dins la caixa usada. L'ús es fa amb el total desconeixement de com s'ha fet la construcció.

Construcció aïllada. Al seu torn els servidors només s'han de preocupar de ser consistents amb la interfície que exposen: l'interior de la caixa que no s'exposi en la interfície no afecta l'ús que en fan els clients, pel simple fet que aquests no hi poden accedir. Per tant, la construcció es realitza de manera aïllada, sense tenir en compte on i com s'usa la caixa negra.

4.8 Desenvolupament sota contracte amb caixes negres

El contracte com a mecanisme de comunicació. El *desenvolupament sota contracte* amb *caixes negres* ens porta un pas més enllà. No només el client i el servidor es poden desenvolupar de manera independent sinó que a més la doble confiança que implica el desenvolupament sota contracte simplifica enormement els desenvolupaments. L'únic punt de contacte entre el client i el servidor, i que per tant cal mantenir estable, és el contracte ofert pel servidor. El contracte *comunica* al client què s'espera d'ell quan usi la caixa negra; el contracte *comunica* al servidor què ha d'oferir al client sota les condicions adients.

Desenvolupament independent. Per desenvolupar el client no cal disposar del servidor, n'hi ha prou en usar-lo sota les condicions del seu contracte (és a dir, cal assegurar les PRE). Per desenvolupar un servidor no cal disposar de cap client; n'hi ha prou en assumir les condicions correctes d'ús (les PRE) i assegurar les condicions exigides pel contracte (les POST).

Algunes conseqüències. El desenvolupament sota contracte amb caixes negres permet fàcilment el desenvolupament descendent i el modular, entre d'altres: de les unitats o mòduls usats l'únic que cal explicitar és el contracte. També permet fàcilment la divisió de tasques: els contractes es converteixen en els punts de comunicació entre els diferents equips desenvolupadors.

5 Regles i principis

> Els principis són recomanacions. Les regles obligacions.

En el disseny regnen els principis. En el disseny treballem amb principis. Per tant, en recomanacions que podem seguir o no, i que fins i tot en alguns casos poden esdevenir contradictòries.

Regles en el disseny. Alguns principis, però, els considerarem d'obligat compliment. Concretament, només considerem *caixes negres*; exigirem que aquestes explicitin un *contracte*; i usarem els contractes per dividir les responsabilitats del client i el servidor (principi de *Desenvolupament sota contracte*). És a dir, el disseny exigirà l'obligat compliment de les següents pautes i principis:

- Caixa negra com a únic tipus d'unitat
  - Encapsulament
  - Ocultació total
  - Interfície obligada
- Contracte explícit[7]
  - Desenvolupament sota contracte

[7] Usarem contracte PRE/POST

6 Principis i definicions

Principis

Definicions

7 Referències

La presentació dels principis de desenvolupament en la bibliografia generalment va lligada a mètodes concrets de desenvolupament o a tecnologies específiques d'implementació. Per aquest motiu es fa difícil fer un recull de referències que presentin els principis de desenvolupament en termes més abstractes.

L'elecció realitzada consta de referències que plantegen el paper dels models en el desenvolupament del software, i de referències que, al nostre entendre, fan alguna incursió en en la presentació de principis de manera independent del mètode o de la tecnologia de desenvolupament. Hem fugit de les referències que tracten mètodes o tecnologies concretes.

La bibliografia recent consultada emfasitza en els diferents mètodes de desenvolupament i passa per alt els principis subjacents a qualsevol mètode; o bé planteja patrons de disseny i d'especificació, novament basats en uns principis subjacents implícits; per aquest motiu les referències presentades són força antigues.

[Bec+01] Kent Beck et al. *Manifesto for Agile Software Development*. 2001. URL: http://agilemanifesto.org/.

[Bud03] D. Budgen. *Software Design*. International Computer Science Series. Pearson/Addison-Wesley, 2003. ISBN: 9780201722192.

[Fow96] Martin Fowler. *Analysis Patterns: Reusable Object Models*. Addison-Wesley, 1996. ISBN: 978-0-201-89542-1.

[FRF02] Martin Fowler, David Rice, and Matthew Foemmel. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002. ISBN: 0321127420.

[Mar02] Robert C. Martin. *Agile Software Development, Principles, Patterns, and Practices*. 1st. Alan Apt Series. Upper Saddle River, NJ: Prentice Hall, Oct. 25, 2002. ISBN: 0135974445.

[Mey97] Bertrand Meyer. *Object-Oriented Software Construction*. 2. Upper Saddle River, NJ: Prentice Hall, 1997. ISBN: 978-0-13-629155-8.

[Mey+98] B. Meyer et al. *Construcción de software orientado a objetos*. Prentice Hall, 1998. ISBN: 9788483220405.

[MO92] J. Martin and J.J. Odell. *Object-oriented analysis and design*. James Martin books on information systems. Prentice Hall, 1992. ISBN: 9780136302452.

[MO96] James Martin and James Odell. *Object-oriented methods: pragmatic considerations*. Prentice-Hall, Inc., 1996. ISBN: 0-13-630864-3.

[MO98] James Martin and James J. Odell. *Object-oriented methods (UML ed., 2nd ed.): a foundation*. Prentice-Hall, Inc., 1998. ISBN: 0-13-905597-5.

[Ode97] James J. Odell. *Advanced object-oriented analysis and design using UML*. SIGS Books, 1997, pp. I–XIV, 1–246. ISBN: 978-0-521-64819-6.

[Ode98] James J. Odell. *Advanced Object-Oriented Analysis and Design Using UML*. Cambridge University Press, 1998. ISBN: 978-0-521-64819-6.

[WM03] R. Wirfs-Brock and A. McKean. *Object Design: Roles, Responsibilities, and Collaborations*. Addison-Wesley Object Technologiey Series. Addison-Wesley, 2003. ISBN: 9780201379433.